设计文档

1.设计综述

P7实现的是一个响应中断的复杂流水线。

指令集

- Add, Sub, And, Or, Slt, Sltu, Lui;
- Addi, Andi, Ori;
- Lb, Lh, Lw, Sb, Sh, Sw;
- Mult, Multu, Div, Divu, Mfhi, Mflo, Mthi, Mtlo;
- Beq, Bne, Jal, Jr;
- Mfc0, Mtc0, Eret, Syscall;
- Subu, Addu, Addiu;（额外实现）

1.1.CPU

与P6不同的是：产生异常并响应；添加指令。

1.1.1.CU

端口说明

输入信号	方向	描述
Instr[31:0]	I	指令
opcode[5:0]	I	Instr[31:26]
func[5:0]	I	Instr[5:0]

控制信号	方向	描述
NPCOp[3:0]	O	NPC的控制信号
CMPOp[2:0]	O	CMP的控制信号
RFWrEn	O	RF的写使能
RFWRSel[2:0]	O	RF写寄存器的选择信号
RFWDSel[2:0]	O	RF写数据的控制信号
EXTOp[1:0]	O	EXT的控制信号
Start	O	指令是否为四种乘除信号之一
MDUOp[3:0]	O	MDU的控制信号
ALUOp[3:0]	O	ALU的控制信号
ALUASel[1:0]	O	ALUA的选择信号
ALUBSel[1:0]	O	ALUB的选择信号
CP0WrEn	O	CP0的写使能
DMWrEn	O	DM的写使能
DMOp[1:0]	O	DM的控制信号
DMEXTOp	O	DM的符号扩展控制信号

阻塞/转发	方向	描述
calc_r	O	指令为“ALU计算&R”型
calc_i	O	指令为“ALU计算&I”型
lui	O	指令为“Lui”型
md	O	指令为“乘除法”型
mt	O	指令为“写HILO型”
mf	O	指令为“读HILO且写寄存器”型
load	O	指令为“Load”型
store	O	指令为“Store”型
branch	O	指令为“Branch”型
j_l	O	指令为“跳转-链接”型
jr	O	指令为“跳转至寄存器”型
mfc0	O	指令为“读CP0且写寄存器”型
mtc0	O	指令为“写CP0”型
eret	O	指令为“eret”型

异常	方向	描述
isAri	O	溢出非法的计算指令
isLw	O	Lw指令
isLh	O	Lh指令
isLb	O	Lb指令
isLoad	O	Load型指令
isSw	O	Sw指令
isSh	O	Sh指令
isSb	O	Sb指令
isStore	O	Store型指令
isSyscall	O	Syscall指令
isRI	O	无定义的非法指令

1.1.2.Stall

组成：异常阻塞（`Stall_eret`），乘除槽阻塞（`Stall_md`），寄存器阻塞（`Stall_rt` / `Stall_rs`）

```
Stall = Stall_eret || Stall_md || Stall_rt || Stall_rs;
```

异常阻塞Stall_eret

当E / M级正在执行写CP0的EPC，且，D级为eret指令，阻塞。

```
wire Stall_eret = (D_Eret) && ((E_Mtc0 && E_rd == 5'd14) || (M_Mtc0 && M_rd
== 5'd14));
```

乘除槽阻塞Stall_md

当Busy或Start为1时，乘除法正在进行且D级指令将读写HILO，阻塞。

```
wire Stall_md = ((Busy||Start)&&(D_Mt||D_Mf)) ? 1 : 0;
```

寄存器阻塞Stall_rt / Stall_rs

- $T_{new}>T_{use}$

> **需求者的最晚时间模型·T_{use}**
>
> T_{use}：指令进入D级后，其后的某个功能部件再经过多少cycle就**必须**要使用寄存器值

> **供给者的最早时间模型·T_{new}**
>
> T_{new}：位于E级及其后各级的指令，再经过多少cycle能够产生要写入寄存器的结果。

- 用的寄存器和写的寄存器是同一个，即D_rs_addr==A3
- 用的寄存器不为$0

  真值表

  - T_{use}：无定义处应取极限大值。避免stall误触发。

指令	T_{use}^{rs}	T_{use}^{rt}
calc_r	1	1
calc_i	1	3
md	1	1
mt	1	3
load	1	3
store	1	2
branch	0	0
jr	0	3
mtc0	3	2

  - T_{new}：无定义处应取0。避免stall误触发。

指令类型	T_{new}@E	T_{new}@M	T_{new}@W
calc_r	1	0	0
calc_i	1	0	0
mf	1	0	0
load	2	1	0
mfc0	2	1	0

端口说明

信号名称	方向	描述
D_Instr	I	D级指令
E_Instr	I	E级指令
M_Instr	I	M级指令
E_A3	I	E级指令写的寄存器
M_A3	I	M级指令写的寄存器
Start	I	乘除法指令开始标志
Busy	I	乘除法正在进行的标志
Stall	O	高电平暂停

1.1.3.F_PC

端口说明

信号名称	方向	描述
WE	I	写使能(!Stall)
req	I	中断异常请求
clk	I	时钟信号
reset	I	同步复位
NPC	I	来自NPC的次地址
PC	I	更新后的地址(`temp_F_PC`)

特殊处理

输出的`temp_F_PC`并非真正的F_PC结果。因为`eret`指令无延迟槽，所以`F_PC`应根据`eret`的真值确定结果为`EPC`还是`temp_F_PC`。

```
F_PC = D_Eret ? EPCOut : temp_F_PC;
```

1.1.4.FD_REG

端口说明

信号名称	方向	描述
reset	I	同步复位
req	I	中断异常请求
WE	I	写使能(!Stall)
flush	I	高电平时清空延迟槽。flush=CMPOp==`CMPOp_Bxxzall && !D_b_jump
F_isBD	I	延迟槽指令标记（D_Branch||D_J_I||D_Jr）
D_isBD	O	
F_ExcCode	I	异常码
FD_ExcCode	O	流水来的前级异常码
F_Instr	I	F级指令
D_Instr	O	D级指令
F_PC	I	F级地址
D_PC	O	D级地址
clk	I	时钟信号

特殊说明

优先级：reset > req > flush / stall

reset：全部清零

req： PC置异常处理程序入口地址。延迟槽标记清零。

WE && flush：清空延迟槽。PC置0，延迟槽标记**保留**。

WE：普通流水。对指令：若前级指令异常，指令置为nop；否则正常流水。

1.1.5.D_CU

控制`NPC`，`RF`，`EXT`，`CMP`。

需要得到`RFWRSel`，以确定`D_A3`，才能判断转发。`A3`参与流水。

`WD`不参与流水，因为它具有不确定性，故让其`MUX`数据集进入流水。

端口说明

信号名称	方向	描述
Instr[31:0]	I	D_Instr
opcode	I	D_Instr[31:26]
func	I	D_Instr[5:0]
NPCOp	O	NPC的控制信号
RFWRSel	O	RF-A3的选择信号
EXTOp	O	EXT的控制信号
CMPOp	O	CMP的控制信号
isSyscall	O	Syscall指令
isRI	O	无定义的非法指令

特殊说明-异常码

在D级判断`Syscall`和`RI`异常。

1.1.6.D_NPC

当D指令的NPCOp为顺序default值时，`NPC=F_PC+4`；当D指令的`NPCOp`为B类时，`NPC`采用`PC@F+4`；当D指令为JL类，NPC只与**D**相关；当D指令为JR类时，`NPC = RS`；当`req=1`时，NPC为`0x4180`；当`NPCOp`为`Eret`时，`NPC = EPC+4`

输入RS被转发处理过。

NPC输出**直接回写F级。**

端口说明

信号名称	方向	描述
req	I	中断异常请求
EPC[31:0]	I	EPC
F_PC	I	F级指令
NPCOp	I	D级指令对应的NPC控制信号
IMM26	I	26位立即数
IMM16	I	16位立即数
RS	I	jr需要的寄存器内容
Branch	I	B指令跳转选择
NPC	O	回写F级IFU

控制信号

NPCOp	描述
NPC_Eret	eret指令
NPC_PC4	顺序
NPC_Br	分支
NPC_JL	跳转并链接
NPC_JR	跳转寄存器内容

1.1.7.D_RF

只实例化一次，D级只使用读功能。写功能的部署在W级。

回写：操作在RF中进行，输出在`mips.txt`中进行。

- **内部转发**：当读和写同一个寄存器时，**读出的数据应该为要写入的数据。**
- 输出D_V1和D_V2**经过转发处理，或许进入流水。**

端口说明

信号名称	方向	描述
clk	I	时钟信号
reset	I	同步复位
A1	I	rs寄存器编号，D_Instr[25:21]
A2	I	rt寄存器编号，D_Instr[20:16]@D
RD1	O	D_V1
RD2	O	D_V2
W_Instr	I	W级指令
RFWrEn	I	W级指令控制
A3	I	W_A3
WD	I	W_RW

1.1.8.D_EXT

输出D_E32进入流水。

端口说明

信号名称	方向	描述
IMM16	I	16位立即数
EXTOp	I	EXT的控制信号
IMMEXT	O	32位扩展立即数D_E32 / lui结果

控制信号

EXTOp	描述
EXTOp_zero	0扩展
EXTOp_sign	符号扩展
EXTOp_lui	lui计算

1.1.9.D_CMP

端口说明

信号名称	方向	描述
D1	I	经过转发处理的GRF[rs]，即FD_rs
D2	I	经过转发处理的GRF[rt]，即FD_rt
CMPOp	I	CMP的控制信号
Branch	O	NPC的跳转信号

控制信号

CMPOp	描述
CMPOp_bep	beq比较
CMPOp_bne	bne比较

1.1.10.DE_REG

输出E_A3进入流水，E_V2经过转发处理，或许进入流水。

控制信号

信号名称	方向	描述
reset	I	同步复位
req	I	中断异常请求
WE	I	写使能，默认为1
flush	I	阻塞时插入nop（Stall）
D_isBD	I	延迟槽标记
E_isBD	O	
D_ExcCode	I	异常码
DE_ExcCode	O	前级流水的异常码
D_b_jump	I	D级B指令是否跳转
E_b_jump	O	
D_V1	I	经过转发处理的GRF[rs]，即FD_rs
D_V2	I	经过转发处理的GRF[rt]，即FD_rt
D_E32	I	32位扩展立即数 / **lui的计算结果**
D_Instr	I	指令
D_PC	I	地址
D_A3		写寄存器编号
E_Instr	O	
E_PC	O	
E_V1	O	GRF[rs]
E_V2	O	GRF[rt]
E_E32		32位扩展立即数 / **lui的计算结果**
E_A3		
clk	I	时钟信号

特殊说明

reset：全部置0

req：PC置异常处理程序入口地址，延迟槽标记清零

flush（Stall）：PC仍然流水。延迟槽标记也流水。

WE：对于指令：若前级异常，则置为nop；否则正常流水。

1.1.11.E_ALU

输出AO进入流水。

端口说明

信号名称	方向	描述
isALUOv	O	计算溢出标志
A	I	经过转发处理和MUX选择的操作数A
B	I	经过转发处理和MUX选择的操作数B
ALUOp	I	ALU的控制信号
ALU_RES	O	ALU的结果

控制信号

ALUOp	描述
ADD	加法
SUB	减法
OR	或运算
AND	与运算
SLT	有符号比较置一
SLTU	无符号比较置一

1.1.12.E_MDU

HI和LO、ALU_RES用三目运算符，合流进E_AO。同时改ALU输出为E_ALU_RES。

端口说明

信号名称	方向	描述
req	I	中断异常请求
A	I	GRF[rs]
B	I	GRF[rt]
LO	O	乘法：低32位；除法：商
HI	O	乘法：高32位；除法：余数
Start	I	指令是否为四种乘除之一
Busy	O	高电平：正在进行乘除法
clk	I	时钟信号
reset	I	同步复位
MDUOp	I	MDU的控制信号

特殊说明

指令**正要开始计算**时，若`req=1`，不进行；否则进行。即计算使能条件加上`&& !req`

1.1.13.EM_REG

端口说明

信号名称	方向	描述
reset	I	同步复位
req	I	中断异常请求
flush	I	刷新信号（0）
WE	I	写使能（1）
E_isBD	I	延迟槽标记
M_isBD	O	
E_ExcCode	I	异常码
EM_ExcCode	O	前级流水的异常码
E_b_jump	I	E级B指令是否跳转
M_b_jump	O	
E_V2	I	经过转发处理的GRF[rt]，即FE_rt
E_AO	I	ALU的结果
E_Instr	I	指令
E_PC	I	地址
E_A3	I	写寄存器编号
M_V2	O	
M_AO	O	
M_Instr	O	
M_PC	O	
M_A3	O	
clk	I	时钟信号

特殊说明

reset：全部清零

req：PC置异常处理程序入口地址，BD其实无所谓。(但先写成流水)

WE：对指令：前级异常则nop，否则正常流水。

1.1.14.M_BE

`req`和`WE`共同作用。即真正的`WrEn = WE && !req`

SJudge和LJudge的工作：

- SJudge：根据DMOp，决定DMByteEn高电平位和WD有效位。注意：DMByteEn的第`i`位为高电平=m_data_rdata的第`i`个字节写入fixed_data的第`i`个字节。这和`sb`、`sh`的定义不太一样，所以要将生数据处理，将有效位移到有效字节范围。
- LJudge：根据DMOp和DMEXTOp，将DM读出的生数据进行处理，后输出。

端口说明

信号名称	方向	描述
req	I	中断异常请求
WE	I	写使能（Store指令触发，作用域：SJudge）
WD_raw	I	未经处理的DM写数据(FM_DW)
WD	O	经过处理的DM写数据（m_data_rdata）
DMRD_raw	I	DM读出的32位生数据
DMAddr	I	DM写地址
DMOp	I	字节操作的控制信号
DMEXTOp	I	字节符号扩展的控制信号
DMByteEn	O	字节使能
DMRD	O	经过字节操作和符号扩展的数据

1.1.15.MW_REG

`reset || req`：全部清零

其他：正常流水

端口说明

信号名称	方向	描述
reset	I	同步复位
req	I	中断异常请求
flush	I	刷新（0）
WE	I	写使能（1）
M_CP0Out	I	CP0读出数据
W_CP0Out	O	
M_b_jump	I	M级B指令是否跳转
W_b_jump	O	
M_AO	I	ALU计算出的结果
M_DR	I	DM读出的结果
M_Instr	I	指令
M_PC	I	地址
M_A3	I	写寄存器编号
W_AO	O	
W_DR	O	
W_Instr	O	
W_PC	O	
W_A3		
clk	I	时钟信号

Forward

选择数据

借用mips.v中的cu进行译码，确定供给者是否转发（信号：Forward）。

采用**就近原则**选择级次转发的数据。

转发输出(优先级：高->低)	恰在本级产生寄存器结果的指令	供给者
DE_REG	J_L / LUI	E_PC+32'd8 / E_E32
EM_REG	CALC_R / CALC_I / Mf	M_PC+32'd8 / M_AO
MW_REG	LOAD / MFC0	W_RW
寄存器内部转发		

1.2.CP0

- 位置：M级
- 实现3个寄存器：`SR(5'd12)`, `Cause(5'd13)`, `EPC(5'd14)`
- 写寄存器：时序逻辑。时序逻辑下有两种写的情况：`mtc0`和请求
- `ExcCode`：**中断优先级高于异常**，故

```
1 `ExcCode <= (IntReq) ? 5'b0 : ExcCodeIn ;
```

端口说明

信号名称	方向	描述
clk	I	
reset	I	
WE	I	CP0写使能，由mtc0驱动
CP0Addr	I	读/写寄存器地址
CP0In	I	CP0寄存器写数据
CP0Out	O	CP0寄存器读出数据
VPC	I	受害PC
BDIn	I	当前指令为延迟槽指令
ExcCodeIn	I	当前异常类型
HWInt[5:0]	I	当前外部中断`{3'b0, interrupt, TC1_IRQ, TC0_IRQ}`
EXLClr	I	EXL清零信号，由eret驱动
EPCOut	O	EPC寄存器内容
Req	O	请求进入处理程序
IG_Response	O	响应中断发生器的标志

寄存器信息：SR(12)-配置异常的功能, Cause(13)-记录异常发生的原因和情况, EPC(14)-记录异常处理结束后需要返回的PC

当发生异常时，CPU自动将异常信息写入CP0的相应寄存器。异常处理程序会访问响应寄存器以了解异常的信息，进行异常处理。

寄存器	功能域	位域	解释
SR	IM	15:10	位域分别对应6个外部中断。1-允许中断；0-禁止中断。被动功能，只能通过`mtc0`指令进行修改。
SR	EXL	1	任何异常发生时置位->强制进入异常处理程序（核心态），并禁止中断
SR	IE	0	全局中断使能。1-允许中断；0-禁止中断
Cause	BD	31	1-EPC指向当前指令的前一条指令；0-指向当前指令
Cause	IP	15:10	为 6 位**待决**的中断位，分别对应 6 个外部中断，1-有中断；0-无中断。每个周期被修改一次，修改内容来自计时器和外部中断。
Cause	ExcCode	6:2	**流水到CP0所在流水级。**异常编码，记录当前异常内容
EPC			记录异常处理结束后需要返回的PC。若异常指令是延迟槽指令，那么存的是异常指令的PC-4，否则存异常指令的PC

异常码

异常与中断码	助记符与名称	指令与指令类型	描述
0	Int(外部中断)	所有指令	中断请求，来源于计时器与外部中断
4	AdEL(取指异常)	所有指令	PC地址未字对齐
4	AdEL(取指异常)	所有指令	PC地址超过 `0x3000 ~ 0x6ffc`
4	AdEL(取数异常)	lw	取数地址未与4字节对齐
4	AdEL(取数异常)	lh	取数地址未与2字节对齐
4	AdEL(取数异常)	lh，lb	取Timer寄存器的值
4	AdEL(取数异常)	load型	计算地址时加法溢出
4	AdEL(取数异常)	load型	取数地址超出DM, Timer0, TIMEr1, 中断发生器的范围
5	AdES(存数异常)	sw	存数地址未4字节对齐
5	AdES(存数异常)	sh	存数地址未2字节对齐
5	AdES(存数异常)	sh，sb	存Timer寄存器的值
5	AdES(存数异常)	store型	计算地址时加法溢出
5	AdES(存数异常)	store型	向计时器的count寄存器存值
5	AdES(存数异常)	store型	存数地址超出DM, Timer0, TIMEr1, 中断发生器的范围
8	Syscall	syscall	系统调用
10	RI	-	未知指令码
12	Ov	add，addi，sub	算术溢出

1.3.Bridge

m_int_addr 和 m_int_byteen在mips.v中实现：具体实现方式记得看助教回复

端口说明

信号名称	方向	描述
CPU_data_addr	I	
CPU_data_wdata	I	
CPU_data_byteen	I	
CPU_data_rdata	O	
m_data_addr	O	
m_data_wdata	O	
m_data_byteen	O	
m_data_rdata	I	
TC0_Addr	O	
TC0_WE	O	
TC0_Din	O	
TC0_Dout	I	
TC1_Addr	O	
TC1_WE	O	
TC1_Din	O	
TC1_Dout	I	

1.4.TC

信号名称	方向	描述
clk	I	
reset	I	
WE	I	
Addr[31:2]	I	
Din	I	
Dout	O	
IRQ	O	

2.测试方案

3.思考题

1. 请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的?

   键盘、鼠标这类低速设备是通过中断请求的方式进行IO操作的。即当键盘上按下一个按键的时候，键盘会发出一个中断信号，中断信号经过中断控制器传到CPU，然后CPU根据不同的中断信号执行不同的中断响应程序，然后进行相应的IO操作，把按下的按键编码读到寄存器（或者鼠标的操作），最后放入内存中。

2. 请思考为什么我们的 CPU 处理中断异常必须是已经指定好的地址？如果你的 CPU 支持用户自定义入口地址，即处理中断异常的程序由用户提供，其还能提供我们所希望的功能吗？如果可以，请说明这样可能会出现什么问题？否则举例说明。（假设用户提供的中断处理程序合法）

   我认为能。用户输入一个地址，CPU读取后进行处理。但这样会导致CPU的运行效率略降低，因为要等待用户提供中断处理程序。但更重要的是普适性过低。

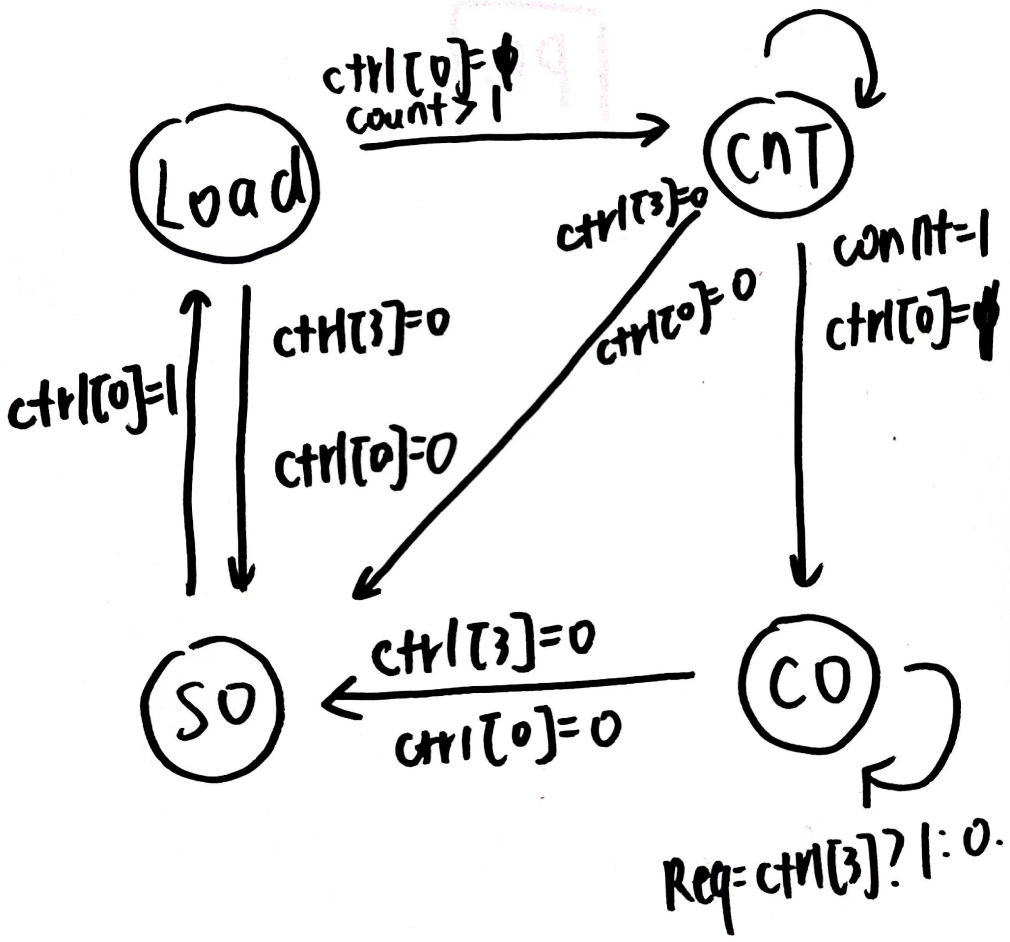
3. 为何与外设通信需要 Bridge?

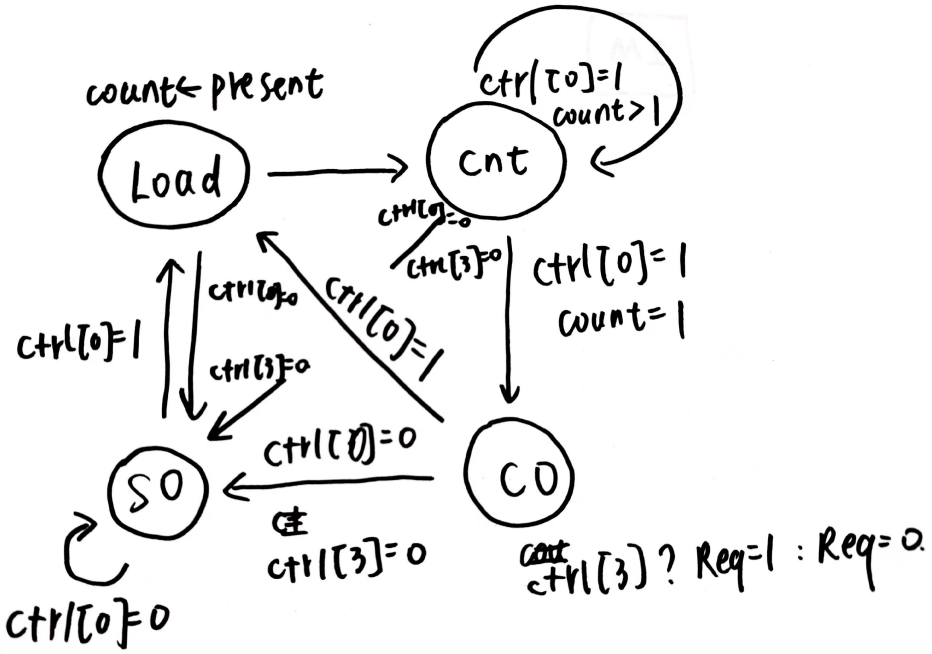
   使得CPU的IO统一化，而不用为每一个外设都添加新的IO口，浪费资源。同时引入Bridge提高了扩展性。

4. 请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并针对每一种模式绘制状态移图。

   同：都从初值寄存器加载初始值。都受控制寄存器的控制。

   异：模式0在计数结束后，一直提供中断。直到使能信号置1或禁止中断，重新加载初值，计数。而模式1计数器每次计数循环中只产生一周期的中断信号，后在条件允许下自动加载初值，计数。





5. 倘若中断信号流入的时候，在检测宏观 PC 的一级如果是一条空泡（你的 CPU 该级所有信息均为空）指令，此时会发生什么问题？在此例基础上请思考：在 P7 中，清空流水线产生的空泡指令应该保留原指令的哪些信息？

   EPC将也为空泡。当`eret`指令作用时，跳入空泡，又产生了新的异常（PC地址超出范围）。从而进入异常死循环。

   故清空流水线产生的空泡指令应保留原指令的PC值和延迟槽标记。

6. 为什么 `jalr` 指令为什么不能写成 `jalr $31, $31`？

   倘若`rs`和`rd`相等，这条指令在重新执行时不会有相同的效果，执行这种指令的结果是UB。即，假如延迟槽指令出现了异常，那么该异常是无法通过重新执行jalr来恢复执行的，因为$31寄存器的值已改变。

7. [P7 选做] 请详细描述你的测试方案及测试数据构造策略。